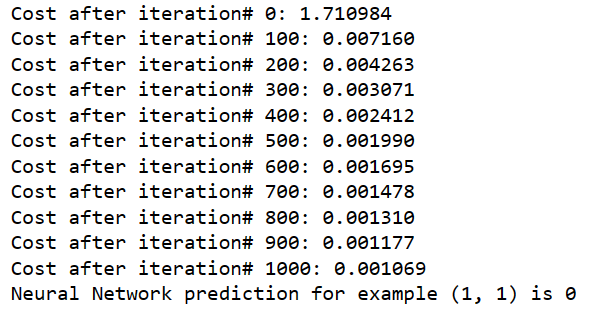
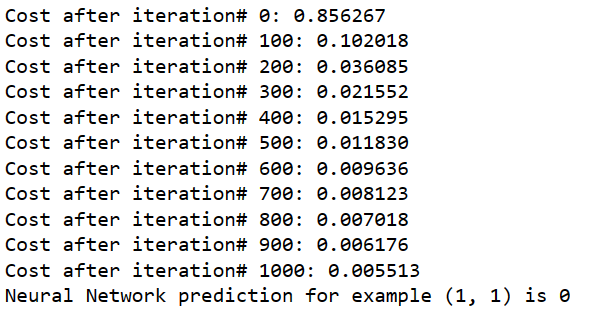
LR=0.3,ITER=1000,NH=2

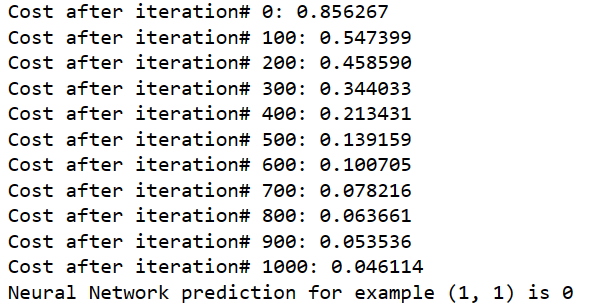
NH=100



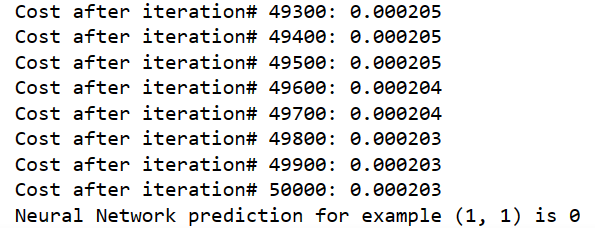
LR=0.6



LR=0.1



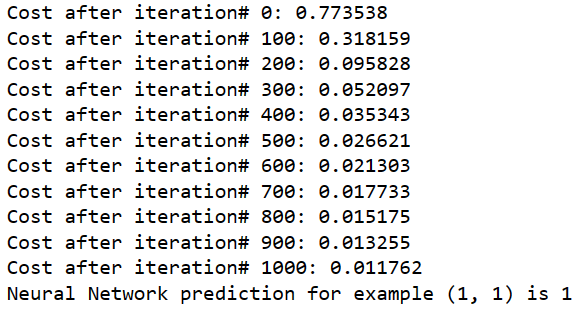
ITER=50000



ITER=50



**XNOR**

****

**import numpy as np**

**def sigmoid(z):**

**return 1/(1 + np.exp(-z))**

**def initialize\_parameters(n\_x, n\_h, n\_y):**

**W1 = np.random.randn(n\_h, n\_x)**

**b1 = np.zeros((n\_h, 1))**

**W2 = np.random.randn(n\_y, n\_h)**

**b2 = np.zeros((n\_y, 1))**

**parameters = {**

**"W1": W1,**

**"b1" : b1,**

**"W2": W2,**

**"b2" : b2**

**}**

**return parameters**

**def forward\_prop(X, parameters):**

**W1 = parameters["W1"]**

**b1 = parameters["b1"]**

**W2 = parameters["W2"]**

**b2 = parameters["b2"]**

**Z1 = np.dot(W1, X) + b1**

**A1 = np.tanh(Z1)**

**Z2 = np.dot(W2, A1) + b2**

**A2 = sigmoid(Z2)**

**cache = {**

**"A1": A1,**

**"A2": A2**

**}**

**return A2, cache**

**def calculate\_cost(A2, Y):**

**cost = -np.sum(np.multiply(Y, np.log(A2)) + np.multiply(1-Y, np.log(1-A2)))/m**

**cost = np.squeeze(cost)**

**return cost**

**def backward\_prop(X, Y, cache, parameters):**

**A1 = cache["A1"]**

**A2 = cache["A2"]**

**W2 = parameters["W2"]**

**dZ2 = A2 - Y**

**dW2 = np.dot(dZ2, A1.T)/m**

**db2 = np.sum(dZ2, axis=1, keepdims=True)/m**

**dZ1 = np.multiply(np.dot(W2.T, dZ2), 1-np.power(A1, 2))**

**dW1 = np.dot(dZ1, X.T)/m**

**db1 = np.sum(dZ1, axis=1, keepdims=True)/m**

**grads = {**

**"dW1": dW1,**

**"db1": db1,**

**"dW2": dW2,**

**"db2": db2**

**}**

**return grads**

**def update\_parameters(parameters, grads, learning\_rate):**

**W1 = parameters["W1"]**

**b1 = parameters["b1"]**

**W2 = parameters["W2"]**

**b2 = parameters["b2"]**

**dW1 = grads["dW1"]**

**db1 = grads["db1"]**

**dW2 = grads["dW2"]**

**db2 = grads["db2"]**

**W1 = W1 - learning\_rate\*dW1**

**b1 = b1 - learning\_rate\*db1**

**W2 = W2 - learning\_rate\*dW2**

**b2 = b2 - learning\_rate\*db2**

**new\_parameters = {**

**"W1": W1,**

**"W2": W2,**

**"b1" : b1,**

**"b2" : b2**

**}**

**return new\_parameters**

**def model(X, Y, n\_x, n\_h, n\_y, num\_of\_iters, learning\_rate):**

**parameters = initialize\_parameters(n\_x, n\_h, n\_y)**

**for i in range(0, num\_of\_iters+1):**

**a2, cache = forward\_prop(X, parameters)**

**cost = calculate\_cost(a2, Y)**

**grads = backward\_prop(X, Y, cache, parameters)**

**parameters = update\_parameters(parameters, grads, learning\_rate)**

**if(i%100 == 0):**

**print('Cost after iteration# {:d}: {:f}'.format(i, cost))**

**return parameters**

**def predict(X, parameters):**

**a2, cache = forward\_prop(X, parameters)**

**yhat = a2**

**yhat = np.squeeze(yhat)**

**if(yhat >= 0.5):**

**y\_predict = 1**

**else:**

**y\_predict = 0**

**return y\_predict**

**np.random.seed(2)**

**# The 4 training examples by columns**

**X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]])**

**# The outputs of the XOR for every example in X**

**Y = np.array([[1, 0, 0, 1]])**

**# No. of training examples**

**m = X.shape[1]**

**# Set the hyperparameters**

**n\_x = 2 #No. of neurons in first layer**

**n\_h = 2 #No. of neurons in hidden layer**

**n\_y = 1 #No. of neurons in output layer**

**num\_of\_iters = 1000**

**learning\_rate = 0.3**

**trained\_parameters = model(X, Y, n\_x, n\_h, n\_y, num\_of\_iters, learning\_rate)**

**# Test 2X1 vector to calculate the XOR of its elements.**

**# Try (0, 0), (0, 1), (1, 0), (1, 1)**

**X\_test = np.array([[1], [1]])**

**y\_predict = predict(X\_test, trained\_parameters)**

**print('Neural Network prediction for example ({:d}, {:d}) is {:d}'.format(**

**X\_test[0][0], X\_test[1][0], y\_predict))**